Knowledge Representation in Entity-Centric Knowledge Bases

Fabian M. Suchanek[1] and Gerhard Weikum[2]

[1] Telecom ParisTech University
[2] Max Planck Institute for Informatics

Abstract. Entity-centric knowledge bases are large collections of facts about entities of public interest, such as countries, politicians, or movies. They find applications in search engines, intelligent assistants, and semantic data mining systems. In this paper, we discuss the knowledge representation that has emerged as a pragmatic consensus in the research community of entity-centric knowledge bases. We describe popular design choices, and point out the assumptions on which they rely. We also discuss current challenges in knowledge representation.

1 Introduction

1.1 Knowledge Bases

When we send a query to Google or Bing, we obtain a set of Web pages. However, in some cases, we also get more information. For example, when we ask "When was Steve Jobs born?", the search engine replies directly with "February 24, 1955". When we ask just for "Steve Jobs", we obtain a short biography, his birth date, quotes, and spouse. All of this is possible because the search engine has a huge repository of knowledge about people of common interest. This knowledge takes the form of a knowledge base (KB).

The KBs used in such search engines are entity-centric: they know individual entities (such as Steve Jobs, the United States, the Kilimanjaro, or the Max Planck Society), their semantic classes (such as *SteveJobs is-a computerPioneer, SteveJobs is-a entrepreneur*), relationships between entities (e.g., *SteveJobs founded AppleInc, SteveJobs hasInvented iPhone, SteveJobs hasWonPrize NationalMedalOfTechnology*, etc.) as well as their validity times (e.g., *SteveJobs wasCEOof Pixar [1986,2006]*).

The vision of a comprehensive KB is not new. It goes back to seminal work in Artificial Intelligence on universal knowledge bases in the 1980s and 1990s, most notably, the Cyc project [19] at MCC in Austin and the WordNet project [10] at Princeton University. These knowledge collections were hand-crafted and manually curated. In the last ten years, automatic knowledge harvesting from Web and text documents has become a major source of information for KBs.

Salient projects with publicly available resources include KnowItAll (UW Seattle, [9]), ConceptNet (MIT, [20]), DBpedia (FU Berlin, U Mannheim, & U Leipzig, [18]), NELL (CMU, [5]), WikiTaxonomy (U Heidelberg & HITS, [22]), WikiData (Wikimedia Foundation, [34]), and YAGO (Telecom ParisTech & Max Planck Institute, [29]). Commercial interest in KBs has been strongly growing, with projects such as the Google Knowledge Graph [8] (including Freebase, [4]), Microsoft's Satori, Amazon's Evi, LinkedIn's Knowledge Graph, and the IBM Watson KB [11]. These KBs contain many millions of entities, organized in hundreds to hundred thousands of semantic classes, and hundred millions of relational facts between entities. Many public KBs are interlinked, forming the Web of Linked Open Data [3].

1.2 Applications

KBs are used in several applications, including the following:

Semantic search and question answering. Both the Google search engine [8] and Microsoft Bing[3] use KBs to give intelligent answers to queries, as we have seen above. They can answer simple factual questions, provide movie showtimes, or show a list of "best things to do" at a travel destination. Wolfram Alpha [4] is another prominent example of a question answering system. The IBM Watson system [11] won against human champions in the TV quiz show Jeopardy, with knowledge from a large KB.

Intelligent assistants. Chatbots such as Apple's Siri, Amazon's Alexa, Google's Allo, or Microsoft's Cortana use KBs to intelligently answer user questions. They can, e.g., suggest restaurants nearby, answer simple factual questions, or manage calendar events. Other companies, too, are experimenting with chat bots that treat customer requests or provide help. With embodiments such as Amazon's Echo system or Google Home, such assistants will share more and more people's homes in the future.

Semantic data mining. Daily news, social media, scholarly publications, and other Web contents are the raw inputs for analytics to obtain insights on business, politics, health, and more. KBs can help to discover and track entities and relationships in order to generate opinion maps, informative recommendations, and other kinds of intelligence towards decision making. For example, we can mine the gender bias from newspapers, because the KB knows the gender of people (see [30] for a survey). There is an entire domain of research (and companies[5]) dedicated to "predictive analytics", i.e., the prediction of events based on past events.

1.3 Knowledge Representation

In this paper, we discuss how the knowledge is usually represented in entity-centric KBs. The field of knowledge representation has a long history, and goes

[3] `http://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing/`
[4] `https://wolframalpha.com`
[5] `https://www.recordedfuture.com/web-intelligence-engine/`

back to the early days of Artificial Intelligence. It has developed numerous knowledge representation models, from frames and KL-ONE to recent variants of description logics. The reader is referred to survey works for comprehensive overviews of historical and classical models [24,26]. In this paper, we discuss the knowledge representation that has emerged as a pragmatic consensus in the research community of entity-centric knowledge bases.

2 Entities

2.1 Entities of Interest

The most basic element of a KB is an *entity*. An entity is any abstract or concrete object of fiction or reality, or, as Bertrand Russell puts it in his *Principles of Mathematics* [36]:

Definition 1 (Entity): *An entity is whatever may be an object of thought.*

This definition is completely all-embracing. Steve Jobs, the Declaration of Independence of the United States, the Theory of Relativity, and a molecule of water are all entities. Events (such as the French Revolution), are entities, too. The notion of an entity is sometimes defined by reference to *existence*. Yet, not all entities in the most general sense exist. Harry Potter, e.g., is an entity, as is phlogiston (the presumed, but inexistant substance that makes up heat).

KBs model a part of reality. This means that they choose certain entities of interest, give them names, and put them into a structure. Thus, a KB is a structured view on part of the world. KBs typically model only distinct entities. This cuts out a large portion of the world that consists of variations, flows and transitions between entities. Drops of rain, for instance, fall down, join in a puddle and may be splattered by a passing car to form new drops [25]. KBs will typically not model these phenomena. Thus, our choice to model only discrete entities is a projection of reality; it is a grid through which we see only distinct things. Many entities consist of several different entities. A car, for example, consists of wheels, a bodywork, an engine, and many other pieces. The engine consists of the pistons, the valves, and the spark plug. The valves consist again of several parts, and so on, until we ultimately arrive at the level of atoms or below. Each of these components is an entity. However, KBs will typically not be concerned with the lower levels of granularity. A KB might model a car, possibly its engine and its wheels, but most likely not its atoms. In all of the following, we will only be concerned with the entities that a KB models.

Entities in the real world can change gradually. For example, the Greek philosopher Eubilides asks: If one takes away one molecule of an object, will there still be the same object? If it is still the same object, this invites one to take away more molecules until the object disappears. If it is another object, this forces one to accept that two distinct objects occupy the same spatio-temporal location: The whole and the whole without the molecule. A related problem is the question of identity. The ancient philosopher Theseus uses the example of a ship: Its old planks are constantly being substituted. One day, the whole ship has been replaced and Theseus asks, “Is it still the same ship?”. To cope with these problems, KBs typically model only atomic entities. In a KB, entities can only be created and destroyed as wholes.

2.2 Identifiers and Labels

In computer systems (as well as in writing of any form), we refer to entities by *identifiers*.

Definition 2 (Identifier): *An identifier for an entity is a string of characters that represents the entity in a computer system.*

Typically, these identifiers take a human-readable form, such as *ElvisPresley* for the singer Elvis Presley. However, some KBs use abstract identifiers. WikiData, e.g., refers to Elvis Presley by the identifier *Q303*, and Freebase by something like */m/012rkqx*. This choice was made so as to be language-independent, and so as to provide an identifier that is stable in time. If, e.g., Elvis Presley reincarnates in the future, then *Q303* will always refer to the original Elvis Presley. It is typically assumed that there exists exactly one identifier per entity in a KB. For what follows, we will not distinguish identifiers from entities, and just talk of entities instead.

Entities have names. For example, the city of New York can be called “city of New York”, “Big Apple”, or “Nueva York”. As we see, one entity can have several names. Vice versa, the same name can refer to several entities. “Paris”, e.g., can refer to the city in France, to a city of that name in Texas, or to a hero of Greek mythology. Hence, we need to carefully distinguish *names* – single words or entire phrases – from their *senses* – the entities that they denote. This is done by using labels.

Definition 3 (Label): *A label for an entity is human-readable string that names the entity.*

If an entity has several labels, the labels are called synonymous. If the same label refers to several entities, the label is polysemous. Not all entities have labels. For example, your kitchen chair is clearly an entity, but does not have any particular label. An entity that has a label is called a *named entity*. KBs typically model mainly named entities. There is one other type of entities that appears in KBs: literals.

Definition 4 (Literal): *A literal is a fixed value that takes the form of a string of characters.*

Literals can be pieces of text, but also numbers, quantities, or timestamps. For example, the label “Big Apple” for the city of New York is a literal, as would be the number of its inhabitants (8,175,133).

3 Classes

3.1 Classes and Instances

KBs model entities of the world. They usually group entities together to form a *class*:

Definition 5 (Class): *A class (also: concept, type) is a named set of entities that share a common trait. An element of that set is called an instance of the class.*

Under this definition, the following are classes: The class of singers (i.e., the set of all people who sing professionally), the class of historical events in Latin America, and the class of cities in Germany. Some instances of these classes are, respectively, Elvis Presley, the independence of Argentina, and Berlin.

Theoretically, KBs can form classes based on arbitrary traits. We can, e.g., construct the class of singers whose concerts were the first to be broadcast by satellite. This class has only one instance (Elvis Presley). We can also construct the class of left-handed guitar players of Scottish origin, or of pieces of music that the Queen of England likes. There are several theories as to whether humans actually build and use classes, too [21]. Points of discussion are whether humans form crisp concepts, and whether all elements of a concept have the same degree of membership. For the purpose of KBs, however, classes are just sets of modeled entities.

It is not always easy to decide whether something should be modeled as an entity or as a class. We could construct, e.g., for every entity a singleton class that contains just this entity (e.g., the class of all Elvis Presleys). Some things of the world can be modeled both as instances and as classes. A typical example is *iPhone*. If we want to designate the type of smartphone, we can model it as an instance of the class of smartphone brands. However, if we are interested in the iPhones owned by different people and want to capture them individually, then *iPhone* should be modeled as a class. A similar observation holds for abstract entities such as *love*. Love can be modeled as an instance of the class *emotion*, where it resides together with the emotions of *anger*, *fear*, and *joy*. However, when we want to model individual love affairs, then *love* would be a class. Its instances are the love affairs that we want to model. In general, both the individual entity *love* and the set of love affairs exist in reality. It is our choice which of the two we add to a KB, which identifier we choose for the instance and which identifier we choose for the class. This is the art of KB modeling.

A pragmatic test of whether something should be modeled as a class is as follows: If we are interested in the plural form of a word or phrase, then we should model it as a class. If we talk, e.g., about "iPhones", then we model several instances of iPhones, and hence *iPhone* should be a class. If we only talk about "iPhone" along with other brand names (such as "HTC One"), then *iPhone* may well be considered an instance. Analogously, if we talk as "love" only in singular, then we may model it as an instance, along with other emotions. If we talk of "loves" (as in "Elvis had many loves during his time as a star"), then *love* is the set of all love affairs – and thus a class. The reason for this test is that only countable nouns can be classes, and only countable nouns can be put into plural. Another method to distinguish classes from instances is to say "An X", or "Every X". If that is possible, then X is best modeled a class, because it can have instances. For example, it is possible to say "a CEO", but not "a

Steve Jobs". Hence, *ceo* should be a class, and *SteveJobs* should not. If we can say "This is X", then X is an instance – as in "This is Steve Jobs". If we can say "X is a Y", then X is an instance of Y – as in "Steve Jobs is a CEO".

A particular case are mass nouns like "milk". The word "milk" (in the sense of the liquid) does not have a plural form. Therefore, we could model it as an instance (e.g., as an instance of the class of liquids). However, if we are interested in individual servings of milk, such as bottles of milk, then we can model it as a class, *servingOfMilk*.

Some KBs do not make the distinction between classes and instances (e.g., the SKOS vocabulary, [39]). In these KBs, everything is an entity. There is, however, usually a "is more general than" link between a more special entity and a more general entity. Such a KB may contain, e.g., the knowledge that *iPhone* is more special than *smartphone*, without worrying whether one of them is a class. The distinction between classes and instances adds a layer of granularity. This granularity is used, e.g., to define the domains and ranges of relations, as we shall see in Section 4.

3.2 Taxonomies

Definition 6 (Subsumption): *Class A is a subclass of class B if A is a subset of B.*

For example, the class of singers is a subclass of the class of persons, because every singer is a person. We also say that the class of singers is a *specialization* of the class of persons, or that singer *is subsumed by* or *included in* person. Vice versa, we say that person is a *superclass* or a *generalization* of the class of singers. Technically speaking, two equivalent classes are subclasses of each other. This is the way the RDFS standard models subclasses [38]. We say that a class is a *proper subclass* of another class, if the second contains more entities than the first. We use the notion of subclass here to refer to *proper* subclasses only.

It is important not to confuse class inclusion with the relationship between parts and wholes. For example, an arm is a part of the human body. That does not mean, however, that every arm is a human body. Hence, *arm* is not a subclass of *body*. In a similar manner, New York is a part of the US. That does not mean that New York would be a subclass of the US. Both New York and the US are instances, so they cannot be subclasses of each other.

Class inclusion is transitive: If A is a subclass of B, and B is a subclass of C, then A is a subclass of C. For example, *vipre* is a subclass of *snake*, and *snake* is a subclass of *reptile*. Hence, by transitivity, *vipre* is also a subclass of *reptile*. We say that a class is a *direct subclass* of another class, if there is no class in the KB that is a superclass of the former and a subclass of the latter. When we talk about subclasses, we usually mean only direct subclasses. The other subclasses are *transitive subclasses*. Since classes can be included in other classes, they can form an inclusion hierarchy – a taxonomy.

Definition 7 (Taxonomy): *A taxonomy is a directed graph, where the nodes are classes and there is an edge from class X to class Y if X is a proper direct subclass of Y.*

The notion of taxonomy is known from biology. Zoological or botanic species form a taxonomy: *tiger* is a subclass of *cat*. *cat* is a subclass of *mammal*, and so on. This principle carries over to all other types of classes. We say, e.g., that *internetCompany* is a subclass of *company*, and that *company* is a subclass of *organization*, etc. Since a taxonomy models proper inclusion, it follows that the taxonomic graph is a acyclic: If a class is the subclass of another class, then the latter cannot be a subclass of the former. Thus, a taxonomy is a directed acyclic graph. A taxonomy does not show the transitive subclass edges. If the graph contains transitive edges, we can always remove them. Given a finite directed acyclic graph with transitive edges, the set of direct edges is unique [27].

Transitivity is often essential in applications. For example, consider a question-answering system where a user asks for artists that are married to actors. If the KB only knew about Elvis Presley and Priscilla Presley being in the classes *rockSinger* and *americanActress*, the question could not be answered. However, by reasoning that *rockSingers* are also *singers*, who in turn are *artists* and *americanActresses* being *actresses*, it becomes possible to give this correct answer.

Usually (but not necessarily), taxonomies are connected graphs: Every node in the graph is, directly or indirectly, linked to every other node. Usually, the taxonomies have a single root, i.e., a single node that has no outgoing edges. This node identifies the most general class, of which every other class is a subclass. In zoological KBs, this may be class *animal*. In a person database, it may be the class *person*. In a general-purpose KB, this class has to be the most general possible class. In YAGO and Wordnet, the class is *entity*. In the RDF standard, it is called *resource* [37]. In the OWL standard [40], the highest class that does not include literals is called *thing*.

Some taxonomies have at most one outgoing edge per node. Then, the taxonomy forms a tree. The biological taxonomy, e.g., forms a tree, as does the Java class hierarchy. However, there can be taxonomies where a class has two distinct direct superclasses. For example, if we have the class *singer* and the classes of *woman* and *man*, then the class *femaleSinger* has two superclasses: *singer* and *woman*. Note that it would be wrong to make *singer* a subclass of *man* and *woman* (as if to say that singers can be men or women). This would actually mean that all singers are at the same time men and women.

When a taxonomy includes a "combination class" such as *FrenchFemaleSingers*, then this class can have several superclasses. *FrenchFemaleSingers*, e.g., can have as direct superclasses *FrenchPeople*, *Women*, and *Singers*. In a similar manner, one entity can be an instance of several classes. Albert Einstein, e.g., is an instance of the classes *physicist*, *vegetarian*, and *violinPlayer*.

When we populate a KB with new entities, we usually try to assign them to the most specific suitable class. For example, when we want to place *Bob Dylan* in our taxonomy, we would put him in the class *americanBluesSinger*, if we have such a class, instead of in the class *person*. However, if we lack more specific information about the entity, then we might be forced to put it into a general class. Some named entity recognizers, e.g., distinguish only between organizations, locations, and people, which means that it is hard to populate more specific classes. It may also happen that our taxonomy is not specific enough at the leaf level. For example, we may encounter a musician who plays the Arabic oud, but our taxonomy does not have any class like *oudPlayer*. Therefore, a class may contain more entities than the union of its subclasses. That is, for a class C with subclasses $C_1, \dots, C_k$, the invariant is $\cup_{i=1..k} C_k \subseteq C$, but $\cup_{i=1..k} C_k = C$ is often false.

3.3 Special Cases

Some KBs assign literals to classes, too. For example, the literal "Hello" can be modeled as an instance of the class *string*. Such literal classes can also form taxonomies. For example, the class *nonNegativeIntegers* is a subclass of the class of *integers*, which is again a subclass of the more general class *numbers*.

Since everything is an entity, a class is an entity as well. Thus, we can construct classes that contain other classes as instances. For example, we can construct the class of all classes *class* $=\{$*car, person, scientist, ...*$\}$. This leads to awkward questions about self-containment, reminiscent of Bertrand Russel's famous set of sets that do not include themselves. The way this is usually solved [37] is to distinguish the class (as an abstract concept) from the extension of the class (the set of its instances). For example, the class of singers is the abstract concept of people who sing. Its extension is the set {Elvis, Madonna, ...}. In this way, a class is not a set, but just an abstract entity. Therefore, the extension of a class can contain another class.

To distinguish classes from other entities, we call an entity that is neither a class nor a literal an *instance* or a *common entity*.

4 Relations

4.1 Relations and Statements

KBs model entities, and place them into classes. They also model *relationships* between entities.

Definition 8 (Relation): *A relationship (also: relation) over the classes* $C_1, ..., C_n$ *is a named subset of the Cartesian product* $C_1 \times ... \times C_n$.

For example, if we have the classes *person*, *city*, and *year*, we may construct the *birth* relationship as a subset of the cartesian product *person*$\times$*city*$\times$*year*. It will contain tuples of a person, their city of birth, and their year of birth. For example, $\langle \mathit{ElvisPresley}, \mathit{Tupelo}, 1935\rangle \in \mathit{birth}$. In a similar manner, we can construct *tradeAgreement* as a subset of *country*$\times$*country*$\times$*commodity*. This relation can contain tuples of countries that made a trade agreement concerning a commodity. Such relationships correspond to classical relations in algebra or databases.

As always in matters of knowledge representation (or, indeed, informatics in general), the identifier of a relationship is completely arbitrary. We could, e.g., call the *birth* relationship *k42*, or, for that matter, *death*. Nothing hinders us to populate the *birth* relationship with tuples of a person, and the time and place where that person ate an ice cream. However, most KBs aim to model reality, and thus use identifiers and tuples that correspond to real-world relationships.

If $\langle x_1, ..., x_n\rangle \in R$ for a relationship R, we also write $R(x_1, ..., x_n)$. In the example, we write $birth(ElvisPresley, Tupelo, 1935)$. The classes of R are called the *domains* of R. The number of classes n is called the *arity* of R. $\langle x_1, ..., x_n\rangle$ is a *tuple of* R. $R(x_1, ..., x_n)$ is called a *statement*, *fact*, or *record*. The elements $x_1, ..., x_n$ are called the *arguments* of the facts. Finally, a *knowledge base*, in its simplest form, is a set of statements. For example, a KB can contain the relations *birth*, *death* and *marriage*, and thus model some of the aspects of people's lives.

4.2 Binary Relations

Definition 9 (Binary Relation): *A binary relation is a relation of arity 2.*

Examples for binary relations are *birthPlace, friendOf*, or *marriedTo*. The first argument of a binary fact is called the *subject*, and the second argument is called the *object* of the fact. The relationships are sometimes called *properties* or *attributes*. Some authors use the term *attributes* only for properties that have literals as objects. The *domain* of a binary relation $R \subset A \times B$ is A, i.e., the class from which the subjects are taken. B is called the *range* of R. For example, the domain of *birthPlace* is *person*, and its range is *city*. The *inverse* of a binary relation R is a relation R^{-1}, such that $R^{-1}(x, y)$ iff $R(x, y)$. For example, the inverse relation of *hasAuthor* is *wroteBook*. The inverse relationship has domain and range swapped.

Any n-ary relation R can be split into n binary relations. This works as follows. Assume that there is one argument position i that is a *key*, i.e., every fact $R(x_1, ..., x_n)$ has a different value for x_i. In the previously introduced 3-ary *birth* relationship, which contains the person, the birth place, and the birth date, the person is the key: every person is born only once at one place. Without loss of generality, let the key be at position $i = 1$. We introduce binary relationships $R_2, ..., R_n$. In the example, we introduce *birthPlace* for the relation between the person and the birth place, and *birthDate* for the relation between the person and the birth year. Every fact $R(x_1, ..., x_n)$ gets rewritten as $R_2(x_1, x_2), R_3(x_1, x_3), R_4(x_1, x_4), ..., R_n(x_1, x_n)$. In the example, the fact *birth(Elvis,Tupelo,1935)* gets rewritten as *birthPlace(Elvis,Tupelo)* and *birthDate(Elvis,1935)*. Now assume that a relation R has no key. As an example, consider again the *tradeAgreement* relationship. Obviously, there is no key in this relationship, because any country can make any number of trade-agreements on any commodity. We introduce binary relationships $R_1, ...R_n$ for every argument position of R. For *tradeAgreement*, these could be *country1*, *country2* and *tradeCommodity*. For each fact of R, we introduce a new entity, an *event entity*. For example, if the US and Brazil make a trade-agreement on coffee, *tradeAgreement(Brazil,US,Coffee)*, then we create *coffeeAgrBrUs*. This entity represents the fact that these two countries made this agreement. In general, every fact $R(x_1, ..., x_n)$ gives rise to an event entity $e_{x1,...,xn}$. Then, every fact $R(x_1, ..., x_n)$ is rewritten as $R_1(e_{x1,...,xn}, x_1), R_2(e_{x1,...,xn}, x_2), ..., R_n(e_{x1,...,xn}, x_n)$. In the example, *country1(coffeeAgrBrUs, Brazil), country2(coffeeAgrBrUs, US), tradeCommodity(coffeeAgrBrUs, Coffee)*. This way, any n-ary relationship can be represented as binary relationships.

The advantage of binary relationships is that they can express facts even if one of the arguments is missing. If, e.g., we know only the birth year of Steve Jobs, but not his birth place, then we cannot make a fact with the 3-ary relation $birth \subset person \times city \times year$. We have to fill the missing arguments, e.g., with *null* values. If the relationship has a large arity, many of its arguments may have to be null values. In the case of binary relationships, in contrast, we can easily state *birthDate(SteveJobs, 1955)*, and omit the *birthPlace* fact. Another disadvantage of n-ary relationships is that they do not allow adding new pieces of information a posteriori. If, e.g., we forgot to declare the astrological ascendant as an argument to the 3-ary relation *birth*, then we cannot add the ascendant for Steve Job's birth. In the binary world, in contrast, we can always add a new relationship *birthAscendant*. Thus, binary relationships offer more flexibility. This flexibility can be a disadvantage, because it allows adding incomplete information (e.g., a birth place without a birth date). However, since knowledge bases are often inherently incomplete, binary relationship are usually the method of choice.

4.3 Functions

Definition 10 (Function): *A function is a binary relation that has for each subject at most one object.*

Typical examples for functions are *birthPlace* and *hasCapital*: Every person has at most one birth place and every country has at most one capital. The relation *ownsCar*, in contrast, is not a function, because a (rich) person can own multiple cars.

Some relations are *functions in time*. This means that the relation can have several objects, but at each point of time, only one object is valid. A typical example is *isMarriedTo*. A person can go through several marriages, but can only have one spouse at a time (in most systems). Another example is *hasNumberOfInhabitants* for cities. A city can grow over time, but at any point of time, it has only a single number of inhabitants. Every function is a function in time.

A binary relation is an *inverse function*, if its inverse is a function. Typical examples are *hasCitizen* (if we do not allow double nationality) or *hasEmailAddress*. Some relations are both functions and inverse functions. They are *bijections*. Bijections are identifiers for objects, such as the social security number. A person has exactly one social security number, and every social security number belongs to exactly one person. Inverse functions and bijections play a crucial role in entity matching: If two KBs talk about the same entity with different names, then one indication for this is that both entities share the same object of an inverse function. For example, if two people share an email address, then the two entities must be identical.

Some relations are “nearly functions”, in the sense that very few subjects have more than one object. For example, most people have only one *nationality*, but some may have several. This idea is formalized by the notion of *functionality* [28]. The functionality of a relation r in a KB is the number of subjects, divided by the number of facts with that relation:

$$fun(r) := \frac{|\{x : \exists y : r(x,y)\}|}{|\{x, y : r(x,y)\}|}$$

The functionality is always a value between 0 and 1, and it is 1 if r is a function.

We usually have the choice between using a relation and its inverse relation. For example, we can either have a relationship *isCitizenOf* (between a person and their country) or a relationship *hasCitizen* (between a country and its citizens). Both are valid choices. In general, KBs tend to choose the relation with the higher functionality, i.e., where the subject has fewer objects. In the example, the choice would probably be *isCitizenOf*, because people have fewer citizenships than countries have citizens. The intuition is that the facts should be “facts about the subject”. For example, the fact that both authors of this paper are citizens of Germany is clearly an important property of the authors (it appears on the Wikipedia page of the second author). Vice versa, the fact that Germany is fortunate enough to count both authors among her citizens is a much less important property of Germany (it does not appear on the Wikipedia page of Germany).

4.4 Relations with Classes

In Section 3.3, we have introduced the class *class*, which contains all classes. This allows us to introduce the relationship between an instance and its class: *type*⊂*entity*×*class*. We can now say *type(Elvis, singer)*. We can also say *type(class, class)*, i.e., *class* is an instance of *class*. We also introduce *subclassOf*⊂*class*×*class*, which is the relationship between a class and its superclasses. For example, *subclassOf(singer, person)*. In the same way as we have introduced the class of all classes, we can introduce the class of all relations. We call this class *property*. With this, we can define the relationship between a binary relation and its domain: *domain*⊂*property*×*class*. We can now say *domain(birthPlace, person)*. Analogously, we introduce *range*⊂*property*×*class*, so that we can say *range(birthPlace, city)*. This way, an entire KB, with its relations and schema information, can be written as binary relationships.

In some cases, we have the choice whether to model something as a relationship or as a class. For example, to say that Berlin is located in Germany, we can either say *locatedIn(Berlin, Germany)* or *type(Berlin, germanCity)*, or both. There is no definite agreement as to which method is the right way to go, but there advantages and disadvantages for each of them. If the entities in question can have certain properties that other entities cannot have, then it is useful to group them into a class. Practically speaking, this means that as soon as there is a relationship that has these entities as domain or range, the entities should become a class. For example, if we model Landkreise (the German equivalent of regions), then we can have *inLandkreis*$\subset$*germanCity*$\times$*Landkreis*. No city other than German cities can be in a Landkreis. Thus, it is useful to have the class *germanCity*. If, however, German cities behave just like all other cities in our KB, then a class for them is less useful. In this spirit, it makes sense to have a class for scientists (who have a graduation university), or digital cameras (which have a resolution), but less so for male scientists or Sony cameras. If we want to express that the entity in question stands in a relationship with another entity, and if that other entity has itself many relationships, then it is useful to use a relational fact. This allows more precise querying. For example, German cities stand in a relationship with Germany. Germany is located in Europe, and it is one of the German speaking countries. Thus, by saying *locatedIn(Berlin, Germany)*, we can query for cities located in European countries and for German-speaking cities, without introducing a class for each of them. In this spirit, it makes sense to use the relational modeling for German cities or American actors, but much less so for, say, zoological categories such as mammals or reptiles. Sometimes neither choice may have strong arguments in favor, and sometimes both forms of modeling together may be useful.

5 Knowledge Bases

5.1 Completeness and Correctness

Knowledge bases model only part of the world. In order to make this explicit, one imagines a complete knowledge base $\mathcal{K}^*$ that contains all entities and facts of the real world in the domain of interest. A given KB $\mathcal{K}$ is *correct*, if $\mathcal{K} \subseteq \mathcal{K}^*$. Usually, KBs aim to be correct. In real life, however, large KBs tend to contain also erroneous statements. YAGO, e.g., has an accuracy of 95%, meaning that 95% of its statements are in $\mathcal{K}^*$ (or, rather, in Wikipedia, which is used as an approximation of $\mathcal{K}^*$). This means that YAGO still contains hundreds of thousands of wrong statements. For most other KBs, the degree of correctness is not even known.

A knowledge base is *complete*, if $\mathcal{K}^* \subseteq \mathcal{K}$ (always staying within the domain of interest). The *closed world assumption* (CWA) is the assumption that the KB at hand is complete. Thus, the CWA says that any statement that is not in the KB is not in $\mathcal{K}^*$ either. In reality, however, KBs are hardly ever complete. Therefore, KBs typically operate under the *open world assumption* (OWA), which says that if a statement is not in the KB, then this statement can be either true or false in the real world. This poses considerable problems, because the absence of a statement does not allow any conclusion about the real world [23].

5.2 The Semantic Web

The common exchange format for knowledge bases is RDF/RDFS [37]. It specifies a syntax for writing down statements with binary relations. Most notably, it prescribes URIs as identifiers, which means that entities can be identified in a globally unique way. RDF is extended by OWL [40], which allows defining also more semantic constraints, such as functions or disjointness of classes, as well as axioms. The formal semantics of these axioms is given by Description Logics [1]. These logics distinguish facts about instances from facts about classes and axioms. The facts about instances are called the *A-Box* ("Assertions"), and the class facts and axioms are called the *T-Box* ("Theory"). Sometimes, the term *ontology* is used to mean roughly the same as *T-Box*. Description Logics allow for automated reasoning on the data.

Many KBs are publicly available online. They form what is known as the *Semantic Web*. Some of these KBs talk about the same entities – with different identifiers. The Linked Open Data project [3] aims to establish links between equivalent identifiers, thus weaving all public KBs together into one giant knowledge graph.

6 Conclusion and Outlook

In this paper, we have discussed the most common way to represent knowledge in today's entity-centric knowledge bases. We have introduced the notions of entities, relations, classes, and statements, and we have discussed the common assumptions on which these notions rely. More generally speaking, knowledge representation is a large field of research, which has received ample attention in the past, and which still harbors many open questions. Some of these open issues in the context of knowledge bases are the following.

Negative Information. KBs usually do not store negative information. They will store that Steve Jobs died, but they will not store that Justin Bieber did not die. This causes problems in conjunction with the open world assumption, because when a statement is absent from the KB, we cannot know whether it is false, or whether it was just not recorded. In some cases, axioms can help deducing negative information. If, e.g., some relation is a function, and if one object is present, then it follows that all other objects cannot be in the relation.

Completeness. Today's KBs do not store the fact that they are complete in some domains. For example, if the KB knows all children of Barack Obama, then it would be helpful to store this knowledge somehow. Different techniques for storing completeness information have been devised (see [23] for a survey), and completeness can also be determined automatically to some degree [13], but these techniques are still in their infancy.

Correctness. Some KBs (e.g., NELL or YAGO) store a probability value with each statement, indicating the likelihood that the statement is correct. There is an ample corpus of scientific work on dealing with such probabilistic knowledge bases, but attaching probabilities to statements is currently not a universally adopted practice.

Provenance. Some KBs (again, e.g., NELL and YAGO) attach provenance information to their statements, i.e., the source where the statement was found, and the technique that was used to extract it. This information can be used to debug the KB, to justify the statements, or to optimize the construction process. Again, there is ample literature on dealing with provenance (see [2] for a survey of works in artificial intelligence, databases, and the Semantic Web) – although few KBs actually attach provenance information.

Time and Space. YAGO attaches time and space information to its facts. Thus, it knows where and when a fact happened. This is achieved by giving each fact a fact identifier, and by making statements about that fact identifier. Other approaches abound [12,24,35,17,16]. They include, e.g., the reification of the statement, the use of 5-ary facts, the introduction of a sub-property for each temporal statement, or the attachment of time labels.

Facts about facts. We sometimes wish to express facts about statements. This can be the time, correctness, or provenance of a fact, as in the previous items, but also the authority who vouches for the fact, access rights to the fact, or beliefs or hypotheses (as in "Fabian believes that Elvis is alive"). RDF provides a mechanism called *reification* for this purpose, but it is clumsy to use. Named Graphs [6] and annotations [33] have been proposed as alternatives. Different other alternatives are surveyed in [2].

Textual extension. The textual source of the facts often contains additional subtleties that cannot be captured in triples. It can therefore be useful to add the textual information into the KB, as it is done, e.g., in [41].

Commonsense knowledge. Properties of everyday objects (e.g. that spiders have eight legs) and general concepts are of importance for text understanding, sentiment analysis, and object recognition in images and videos. This line of knowledge representation is well covered in classical works [24,19], and is lately also enjoying attention in the KB community [31,32].

Intensional knowledge. Commonsense knowledge can also take the form of rules. For example, if a doctoral student is advised by a professor, then the university of graduation will be the employer of the professor. Again, this type of knowledge representation is well covered in classical works [24,19], and recent approaches have turned to using it for KBs [14,15,7].

NoRDF. For some information (such as complex events, narratives, or larger contexts), the representation as triples is no longer sufficient. We call this the realm of NoRDF knowledge (in analogy to NoSQL databases). For example, it is clumsy, if not impossible, to represent with binary relations the fact that Leonardo diCaprio was baptized "Leonardo" by his mother, because she visited a museum in Italy while she was still pregnant, and felt that the baby kicked while she saw a work of Leonardo DaVinci.

We thus conclude that we are still far from being able to represent all relevant knowledge in this world. And yet, the pieces that we can represent are of use already now. They will certainly become more numerous, and more useful, in the future.

References

1. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook*. 2003.
2. Meghyn Bienvenu, Fabian M. Suchanek, and Daniel Deutch. Provenance for Web 2.0 Data . In *SDM workshop*, 2012.
3. Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee. Linked data on the Web. In *WWW*, 2008.
4. Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, 2008.
5. A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. Hruschka Jr., and T. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.
6. Jeremy J Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs, provenance and trust. In *WWW*, 2005.
7. Yang Chen, Daisy Zhe Wang, and Sean Goldberg. ScaLeKB: Scalable Learning and Inference over Large Knowledge Bases . In *VLDBJ*, 2016.
8. X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD*, 2014.
9. Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale information extraction in knowitall:. In *WWW*, 2004.
10. C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.
11. David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3), 2010.
12. Michael David Fisher, Dov M Gabbay, and Lluis Vila. *Handbook of temporal reasoning in artificial intelligence*. 2005.
13. Luis Galárraga, Simon Razniewski, Antoine Amarilli, and Fabian M. Suchanek. Predicting Completeness in Knowledge Bases . In *WSDM*, 2017.
14. Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases . In *WWW*, 2013.
15. Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Fast Rule Mining in Ontological Knowledge Bases with AMIE+ . In *VLDBJ*, 2015.
16. Claudio Gutierrez, Carlos A Hurtado, and Alejandro Vaisman. Introducing time into rdf. *IEEE Transactions on Knowledge and Data Engineering*, 19(2), 2007.
17. Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia . In *Artificial Intelligence* , 2013.

18. Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 6(2), 2015.
19. Douglas B Lenat and Ramanathan V Guha. *Building large knowledge-based systems; representation and inference in the Cyc project.* Addison-Wesley Longman Publishing Co., Inc., 1989.
20. H. Liu and P. Singh. Conceptnet. *BT Technology Journal*, 22(4), October 2004.
21. Eric Margolis and Stephen Laurence. Concepts. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy.* 2014.
22. Simone Paolo Ponzetto and Michael Strube. Wikitaxonomy: A large scale knowledge resource. In *ECAI*, 2008.
23. Simon Razniewski, Fabian M. Suchanek, and Werner Nutt. But What Do We Actually Know? . In *AKBC workshop*, 2016.
24. S. Russell and P. Norvig. *Artificial Intelligence: a Modern Approach.* Prentice Hall, 2002.
25. J. F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations.* Brooks/Cole, 2000.
26. Steffen Staab and Rudi Studer, editors. *Handbook on Ontologies.* International Handbooks on Information Systems. Springer, 2004.
27. F. M. Suchanek. Representing Ontological Structures in CLOS, Java and FAST. B.Sc. Thesis, University of Osnabrück, 2003.
28. Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. PARIS: Probabilistic Alignment of Relations, Instances, and Schema . In *VLDB*, 2012.
29. Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago - A Core of Semantic Knowledge . In *WWW*, 2007.
30. Fabian M. Suchanek and Nicoleta Preda. Semantic Culturomics . In *VLDB short paper track*, 2014.
31. Niket Tandon, Gerard de Melo, Abir De, and Gerhard Weikum. Knowlywood: Mining Activity Knowledge From Hollywood Narratives. In *CIKM*, 2015.
32. Niket Tandon, Gerard de Melo, Fabian M. Suchanek, and Gerhard Weikum. WebChild: Harvesting and Organizing Commonsense Knowledge from the Web . In *WSDM*, 2014.
33. Octavian Udrea, Diego Reforgiato Recupero, and VS Subrahmanian. Annotated rdf. *ACM Transactions on Computational Logic*, 11(2), 2010.
34. Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10), 2014.
35. Chris Welty, Richard Fikes, and Selene Makarios. A reusable ontology for fluents in owl. In *FOIS*, 2006.
36. Alfred North Whitehead and Bertrand Russell. *Principia mathematica.* 1913.
37. Word Wide Web Consortium. RDF Primer, 2004.
38. Word Wide Web Consortium. RDF Vocabulary Description Language 1.0: RDF Schema, 2004.
39. Word Wide Web Consortium. SKOS Simple Knowledge Organization System, 2009.
40. Word Wide Web Consortium. OWL 2 Web Ontology Language, 2012.
41. Mohamed Yahya, Denilson Barbosa, Klaus Berberich, Qiuyue Wang, and Gerhard Weikum. Relationship Queries on Extended Knowledge Graphs. In *WSDM*, 2016.